Interval approximations of message causality in distributed execution : observation d'exécutions réparties datation par intervalles : version anglaise et française

Claire Diehl, Claude Jard

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1571

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

INTERVAL APPROXIMATIONS OF MESSAGE CAUSALITY IN DISTRIBUTED EXECUTION

OBSERVATION D'EXECUTIONS REPARTIES DATATION PAR INTERVALLES

Versions anglaise et française

Claire DIEHL
Claude JARD

Décembre 1991

RR-1571

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Interval Approximations of Message Causality in Distributed Execution

Claire DIEHL, Claude JARD
e-mail: jard@irisa.fr

Publication Interne n°617, Novembre 1991, 44 pages - Programme 1

abstract

In this paper we study timestamping for the dynamic analysis of events ordering in message-passing systems. We use the partial order theory to shed new light on classical timestamping algorithms. We give the basic properties of timestamping and we define the notion of causality order approximation. Consequently we present a new stamp technique based on a special class of order: interval orders. This technique gives better results than the Lamport's logical clocks for the same cost. Lastly we characterize executions for which our timestamping is optimum.

Observation d'Exécutions Réparties Datation par Intervalles

résumé

L'objet de cet article est d'étudier les mécanismes d'estampillage d'exécutions réparties qui permettent une observation *à la volée* des événements. On utilise la théorie des ordres partiels pour donner une nouvelle interprétation des algorithmes d'estampillage classiques. On exprime les propriétés importantes que doit posséder un algorithme de datation et on justifie le fait de n'observer que des approximations de la relation de causalité.
Grâce à la classe des ordres d'intervalles, on met en évidence une nouvelle technique d'estampillage qui, au niveau des performances s'intercale entre les algorithmes existants. Nous caractérisons les exécutions pour lesquelles notre estampillage est optimal.
Cette étude nous montre qu'en restreignant les classes d'exécutions observées et en s'appuyant sur des propriétés connues des ordres partiels, on est en mesure d'observer des approximations des exécutions réparties à un coût constant, c'est à dire non proportionnel à la taille du système.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE (U.R.A. 227) UNIVERSITE DE RENNES I INSA DE RENNES
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE (UNITE DE RECHERCHE DE RENNES)

Interval Approximations of Message Causality in Distributed Execution [1]

Contents

1 The nature of distributed executions **3**
- 1.1 General motivation . . . 3
- 1.2 Events and causality . . . 3
- 1.3 Preliminaries on partial orders . . . 3

2 Timestamping and partial orders **5**
- 2.1 General properties . . . 5
- 2.2 Vector stamping . . . 6
- 2.3 Scalar stamping . . . 7

3 Interval approximations **9**
- 3.1 Improvement of Lamport's algorithm . . . 9
- 3.2 Computations coded by intervals . . . 10
  - 3.2.1 General case . . . 11
  - 3.2.2 The RPC case . . . 12

4 Conclusion and prospects **15**

[1] This report is an article presented at the Workshop STACS'92 in Paris, France, February 1992.

1 The nature of distributed executions

1.1 General motivation

Interest for distributed computing is growing but mastering all its aspects is still a challenge. We need theories and algorithms to handle distributed executions.

In a distributed execution, the ordering of events must be precisely defined. The set of events occurring on a single processor is totally ordered, but the relations between events from distinct processors depends on communication. The observation of this ordering without disturbing the program behavior is still a difficult problem. But this is a necessary step for many applications: diagnosis, performance evaluation, monitoring, replay, animation... Anyway this can help us to have a better understanding of an execution.

As far as possible we will try to stay in a general framework: we consider a distributed system as a collection of n processors which communicate each other by exchanging messages. The number of processors is constant and known. Therefore we confuse the notion of process and the notion of processor. We do not make any assumption about how the computers are connected, provided that the network is connected.

1.2 Events and causality

When a program runs on a group of processors, it defines a set of *actions*. When an action is executed, it is called an *event*. Such an action can be an internal action on a processor, a sending or a receipt of a message to or from another processor. The set of events is ordered by the message causality relation. This order was first presented by Lamport in [Lam78] and is usually called *happened before*. It can be defined as the transitive closure of these two rules: the events happening on a single processor are totally ordered, and the sending of a message occurs before its receipt.

In practice, only particular events, defined as observable, are traced. But all the communicating events are needed to compute the causality relation.

We will use the following notations: θ is the causality relation:

$$\theta(x,y) \iff x \textit{ causally precedes } y \iff$$

$$\textit{there exists a path from } x \textit{ to } y \textit{ in the time diagram of the execution}$$

E is the set of events used for the timestamping and $\mathcal{E} = (E, \theta)$ the associated order. O is the subset of the events which are effectively observed and $\mathcal{O} = (O, \theta)$ the associated suborder.

1.3 Preliminaries on partial orders

We will represent distributed computations using **diagrams**: the transitive reduction of the graph of the observed events ordering where the edge direction is implicit from bottom to top (see figure 1).

The **size** $|\mathcal{E}|$ of an order $\mathcal{E} = (E, \theta)$ is the total number of relations in θ. This is also the transitive closure edge number of the diagram representing $\mathcal{E}$.

A set of pairwise comparable elements is called a **chain** and a set of pairwise incomparable elements is an **antichain**.

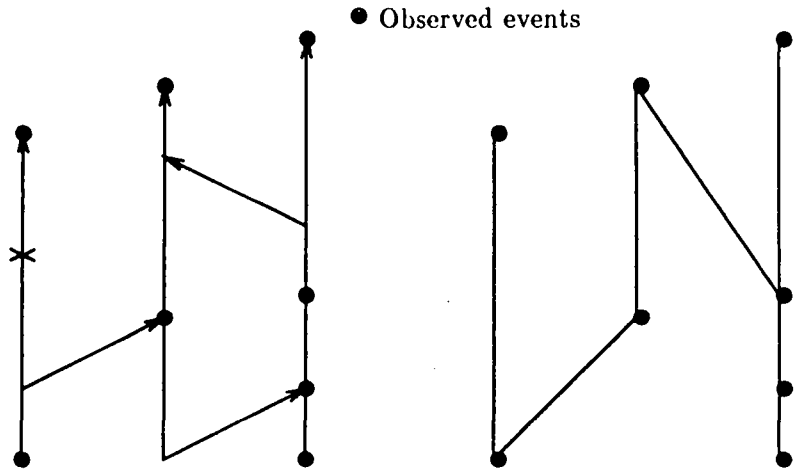

Figure 1: A distributed execution and its associated diagram

The **width** of an order is the length of one of its largest antichains.
y **covers** x, or y is an **immediate successor** of x, and we note $\theta_{im}(x,y)$ when:

$$\theta_{im}(x,y) \Longleftrightarrow \theta(x,y) \text{ and } \nexists z \in E,\ \theta(x,z) \text{ and } \theta(z,y).$$

We define the **predecessor set** for an element $x \in E$ as:

$$Pred_{\mathcal{E}}(x) = \{y \in E / \theta(y,x)\} \text{ and } Pred_{im\mathcal{E}}(x) = \{y \in E / \theta_{im}(y,x)\}$$

In order to describe the stamping mechanisms, we will use the following notations. If e is an event, we denote e^- (resp. e^+) the immediately preceding (resp. following) event on the same processor. If e is the sending of a message, e^r is the receipt of the same message. And if e is a receipt, e^e is the sending of the message (see figure 2).

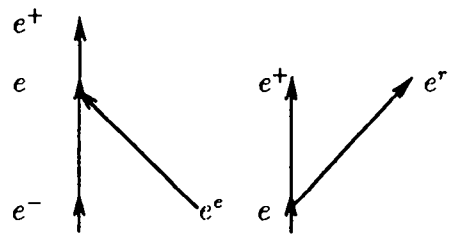

Figure 2: Notations

Interval orders model the sequential structure of intervals of the real line. As there is a large range of applications using the time concept, they have been intensively studied in recent years [Fis85, Moh89, Ram91].

Definition 1 (E,θ) *is an* **interval order** iff *there is a set of intervals* $(I_x)_{x\in E}$ *on the real line such that:*

$$\theta(x,y) \Longleftrightarrow I_x \textit{ is on the left of } I_y$$

This definition gives a nice representation of such an order: we can associate a pair of real numbers to each element. There exist other characterizations of interval orders which can be easily obtained (see for instance [Moh89]):

Theorem 1 *For an order $\mathcal{E} = (E, \theta)$ the following statements are equivalent:*

(i) *$\mathcal{E}$ is an interval order.*

(ii) *$\mathcal{E}$ does not contain any suborder isomorphic to the order "$2 \oplus 2$" (see figure 3).*

(iii) *The maximal antichains of $\mathcal{E}$ can be linearly ordered such that, for every element x the maximal antichains containing x occur consecutively.*

(iv) *The sets of predecessors $Pred_{\mathcal{E}}(x)$ are linearly ordered by inclusion.*

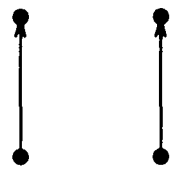

Figure 3: The order "$2 \oplus 2$"

An **extension** of $\mathcal{E} = (E, \theta)$ is an order including θ. A **linear extension** is an extension which totally orders E. An **interval extension** is an extension which is an interval order (see the definition just above).

2 Timestamping and partial orders

2.1 General properties

A timestamping technique associates a date $\delta(x)$ to each event x. If there exists an ordering of the stamps, there also exists a relation between the events.

Coding exactly the causality order of a computation is very expensive in general and can alter it: additional computations, larger messages slow down the execution speed and possibly alter the algorithm behavior. Then we are interested in just observing approximations of the causality order. A timestamping algorithm should have the following properties:

- The stamps ordering extends the causality order.
- The algorithm is incremental: the stamp of an event depends on its preceding events. If necessary it should depend on some events in its future, but this future should be close: in this case the algorithm is said to be *pseudo-incremental.*
- The computation of a stamp on a processor depends on local informations: the algorithm does not have to add any message.

In order to compare two different timestamping algorithms, we will compute the size of the order [CKS86] produced by both algorithms on the same execution.

2.2 Vector stamping

Timestamping by vectors of integers is actually the illustration of a general technique for coding partial orders. **Dilworth theorem** [Dil50] says that an order of width k can be decomposed into k chains. And with such a partition, we can build an order preserving mapping from E to $\mathbb{N}^k$. The following theorem is probably a special case of a general property but the proof helps us to understand the structure of the coding.

Theorem 2 *For $\mathcal{E} = (E, \theta)$ a partially ordered set of width k and $(L_i)_{1 \leq i \leq k}$, a partition into chains of E the map:*

$$\begin{aligned} \phi \; : \; & E \longrightarrow \mathbb{N}^k \\ & x \longmapsto (|(Pred_{\mathcal{E}}(x) \cup \{x\}) \cap L_i|)_{1 \leq i \leq k} \end{aligned}$$

satisfies[2]*:*

$$\forall x, y \in E, \;\; \theta(x,y) \Longleftrightarrow \phi(x) <_{\mathbb{N}^n} \phi(y)$$

Proof: let x and y be such that
$\theta(x,y)$, *i.e.* $x \in Pred_{\mathcal{E}}(y)$.
So $Pred_{\mathcal{E}}(x) \cup \{x\} \subset Pred_{\mathcal{E}}(y) \cup \{y\}$ and clearly
$\forall i \in \{1..k\}$, $\phi(x)_i \leq \phi(y)_i$ and $\exists j \in \{1..k\}$, $y \in L_j$ and $\phi(x)_j < \phi(y)_j$.
Thus we have shown $\Longrightarrow$.
Let x and y be such that
$\forall i \in \{1..k\}$, $|(Pred_{\mathcal{E}}(x) \cup \{x\}) \cap L_i| \leq |(Pred_{\mathcal{E}}(y) \cup \{y\}) \cap L_i|$. Each L_i is a chain, hence
$(Pred_{\mathcal{E}}(x) \cup \{x\}) \cap L_i \subseteq (Pred_{\mathcal{E}}(y) \cup \{y\}) \cap L_i$ and consequently
$Pred_{\mathcal{E}}(x) \cup \{x\} \subset Pred_{\mathcal{E}}(y) \cup \{y\} : \theta(x,y)$. □

When observing a distributed computation, there is a trivial partition of the set of events: each chain L_i contains the events occurring on the i^{th} processor. If there are n processors in the system:

$$E = \biguplus_{1 \leq i \leq n} L_i, \; L_i \text{ contains the events of the } i^{th} \text{ processor.}$$

We denote δ_i the vector whose all components are null except the i^{th} which is equal to 1. The mechanism which constructs the preceding coding of the causality order is the following. Each processor has a clock vector of size n: $\delta(x) \in \mathbb{N}^n$.

- When the first event x of the i^{th} processor occurs, $Pred_{\mathcal{E}}(x) = \emptyset$,
  its clock is initialized to δ_i.
- If the current event $x \in L_i$ is an internal event or the sending of a message,
  $Pred_{\mathcal{E}}(x) = Pred_{\mathcal{E}}(x^-) \cup \{x^-\}$ and therefore
  $\delta(x) = \delta(x^-) + \delta_i$.

- If the current event $x \in L_i$ is the receipt of a message,
  $Pred_{\mathcal{E}}(x) = Pred_{\mathcal{E}}(x^-) \cup Pred_{\mathcal{E}}(x^e) \cup \{x^-, x^e\}$.
  The sets L_j are totally ordered, hence
  $\forall j \in \{1..n\}$, $(Pred_{\mathcal{E}}(x^e) \cup \{x^e\}) \cap L_j \subseteq$ (or $\supseteq$) $(Pred_{\mathcal{E}}(x^-) \cup \{x^-\}) \cap L_j$
  and hence
  $|Pred_{\mathcal{E}}(x) \cap L_j| = max(|(Pred_{\mathcal{E}}(x^e) \cup \{x^e\}) \cap L_j|, |(Pred_{\mathcal{E}}(x^-) \cup \{x^-\}) \cap L_j|)$:
  $\delta(x) = max(\delta(x^-), \delta(x^e)) + \delta_i$.

We recognize the vector clock mechanism proposed in 1988 by Fidge and Mattern [Fid88, Mat89]. This timestamping mechanism codes the causality order: the i^{th} component of an event stamp is the total number of events preceding it on the i^{th} processor and characterizes the predecessors set (see figure 4). To compute this clock, each message has to hold a stamp of length n: this is the main drawback of this algorithm and this becomes unrealistic in large parallel systems. But, as said in [Cha91], the size n is necessary if we look for embeddings of the causality order in $(\mathbb{N}, <_{\mathbb{N}^n})$. This is why we are going to study constant size stamps which will approximate the causality.

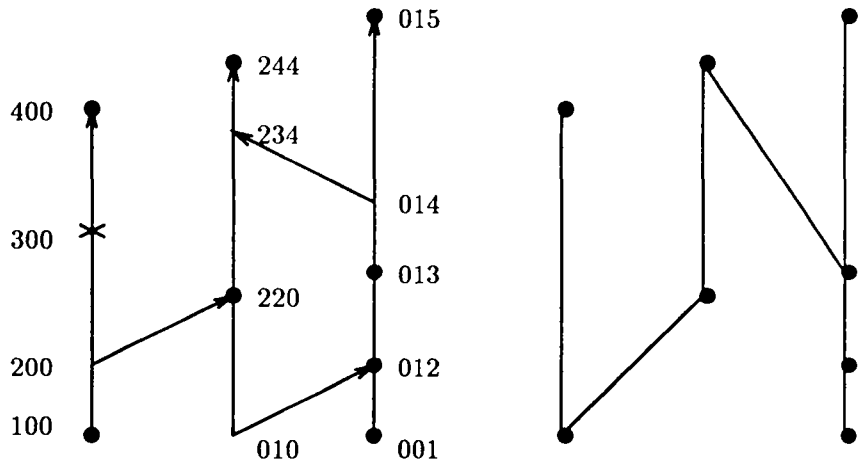

Figure 4: Mattern's vector stamping: $|\mathcal{O}| = 13$

2.3 Scalar stamping

Definition 2 *Let $\mathcal{E} = (E, \theta)$ be a partial order. For $x \in E$ the rank of x, $\rho(x)$ is the maximal length of a chain ending at x.*

The rank concept allows us to partition ordered sets:

Definition 3 *Let $\mathcal{E} = (E, \theta)$ be a partial order over a finite set. The partition:*

$$E = \biguplus_{0 \leq i \leq h} R_i \text{ where } \forall i \geq 0,\ R_i = \{x \in E / \rho(x) = i\}$$

is called the rank decomposition of $\mathcal{E}$.

We prefer the following constructive characterization. It corresponds to a classical breadth-first graph traversal.

Theorem 3 *Let $\mathcal{E} = (E, \theta)$ be a partial order over a finite set. The partition:*

$$E = \biguplus_{0 \leq i \leq h} A_i \text{ where } \begin{array}{l} A_0 = Min(E, \theta) = \{x \in E / \nexists y \in E,\ \theta(y, x)\} \\ \text{and } \forall i > 0,\ A_i = Min(E - \bigcup_{j<i} A_j, \theta) \end{array}$$

*is the rank decomposition (*i.e. *$\forall i,\ R_i = A_i$).*

Proof: let $i \geq 0$ and $x \in A_i$. In view of the definition of the sets A_j we can construct a chain ending at x which contains at least one element in each set $A_j, j < i$. Since it is a chain, it cannot contains two elements from the same A_j (each A_j is an antichain). Thus the length of a maximal chain ending at x is i: $\rho(x) = i$. □

Thus the rank decomposition of a partially ordered set is a partition into antichains. Each antichain is not maximal but the total number of antichains is minimal. This theorem gives us an algorithm which easily computes the rank function. Let $x \in A_i, (i > 0)$, by definition of the sets A_j, at least one immediate predecessor of x is in A_{i-1}. Therefore:

$$\rho(x) = max\{\rho(y),\ \theta_{im}(y, x)\} + 1$$

In particular, when $\mathcal{E} = (E, \theta)$ is the causality order of a distributed execution, the rank is computed by the following rules:

- If e is the first event occurring on a site, except the receipt of a message then e is minimal *i.e.* $e \in A_0$ and $\rho(e) = 0$.
- If e is an internal event or the sending of a message, then $Pred_{im\mathcal{E}}(e) = \{e^-\}$ and $\rho(e) = \rho(e^-) + 1$.
- If e is the receipt of a message then $Pred_{im\mathcal{E}}(e) \subseteq \{e^-, e^e\}$ and $\rho(e) = max(\rho(e^-), \rho(e^e)) + 1$.

This is the same mechanism as the logical clock proposed by Lamport in [Lam78]. Fidge and Mattern's vector clock is based on a chain partition of the event set, the scalar stamp of Lamport corresponds to an antichain decomposition. Figure 5 makes visible the structure of the extension produced by the Lamport's stamp algorithm. This is the structure of a **weak order** : a weak order is a totally ordered set where antichains have been put in place of some elements.

Proposition 1 *The Lamport's timestamping computes the rank of each event for the causality order and the resulting order is a weak order.*

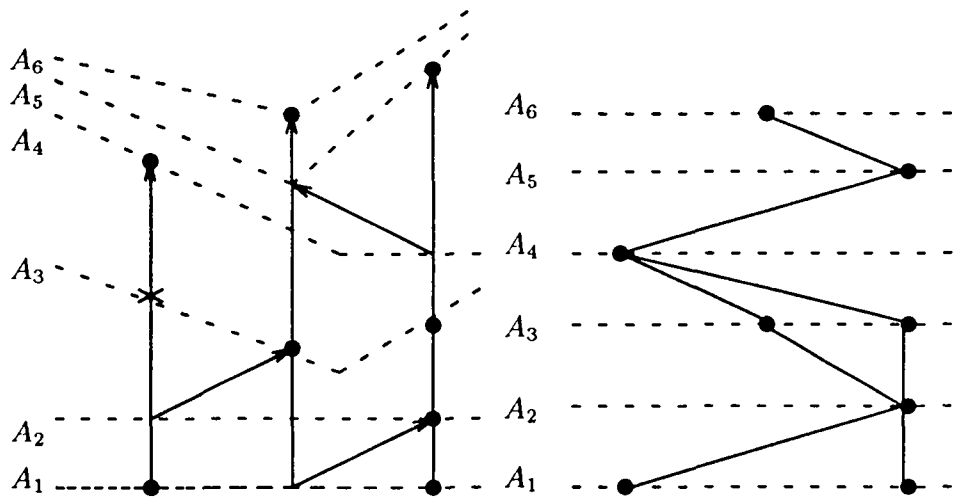

Figure 5: Lamport's scalar stamping:$|\mathcal{O}| = 26$

3 Interval approximations

3.1 Improvement of Lamport's algorithm

The events ordering due to Lamport's clock can be viewed as an interval order. Actually, to each event e, we can associate an interval of the real line: $[\rho(e), \rho(e) + 1[$. The ordering of the intervals is the same as the ordering of the events. Moreover all these intervals have the same length, thus this ordering belongs to a subclass of interval orders called semiorder (a semiorder is an interval order for which there exists a representation with intervals of the same length). The scheme of the figure 6 shows the interval structure of this order.

This remark gives us an idea to improve the Lamport's interval approximation. We associate to each event e the interval $[\delta(e)^-, \delta(e)^+[$ where $\delta(e)^- = \rho(e)$, the Lamport's stamp, and $\delta(e)^+ = min\{\rho(f)/\theta_{im}(e,f)\}$. The value of $\delta(e)^+$ preserves θ because it preserves the ordering of events on a same processor and the causality between a sending and its corresponding receipt. We increase the size of some intervals. Thus it is clear that it will reduce the size of the observed extension.

Proposition 2 *The set* $([\delta(e)^-, \delta(e)^+[)_{e \in E}$ *where:*

$$\delta(e)^- = \rho(e), \textit{ Lamport's stamp}$$
$$\delta(e)^+ = min\{\delta(f)^-/\theta_{im}(e,f)\}$$

gives an interval extension (or an interval approximation) of the causality order.

An illustration of this stamp technique appears in figure 6. In terms of order inclusion this interval approximation is between the message causality and the Lamport's logical clocks.

However this stamp mechanism is no longer incremental: the right interval endpoint does not depend on the past of the events but on a close future. To know the complete stamp of an event, we must wait for all its immediate successors to occur.

- The internal events and the receipts have only one immediate successor: the next event occurring on the same processor. When it occurs we can trace the stamp of the preceding event.
- A message sending e should have two immediate successors: the next event on the same site e^+ and the receipt event e^r. To compute $\delta(e)^+ = min(\delta(e^+)^-, \delta(e^r)^-)$, we trace its two possible values and we take the lowest.

Consequently we have to process the observed trace, but this can be done in parallel with the execution: we do not have to wait for the end of the computation. As we will see in the following sections, there is a better solution to this problem, when all the sendings are followed by a blocking acknowledgment. The disturbance due to the observation is the same than for the Lamport's logical clock: the stamp of each message has the same size (one integer).

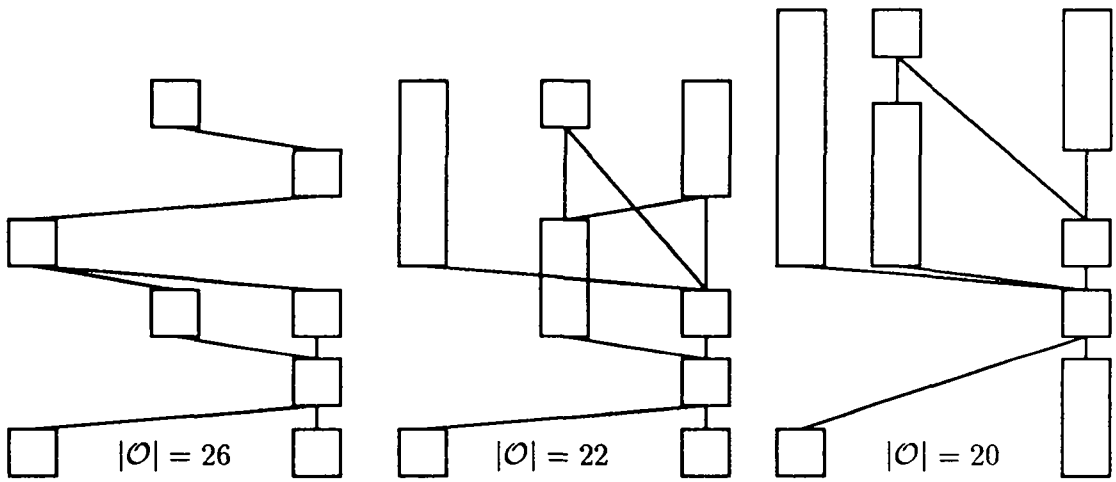

Figure 6: Interval representation of Lamport's stamping, its refinement and the "max+2" stamping

3.2 Computations coded by intervals

In the last section, we have found two interval extensions of the causality order: Lamport's stamping and its refinement. Are there timestamping algorithms which code exactly computations whose causality order is an interval order?

3.2.1 General case

Interval characterization

Papadimitriou and Yannakakis have developed in [PY79] an algorithm for recognizing interval orders. It constructs an interval representation which is founded on the following theorem.

Theorem 4 *Let $\mathcal{E} = (\mathcal{E}, \theta)$ be an interval order. The set of intervals $([\delta(x)^-, \delta(x)^+[)_{x \in E}$ where*[3]*:*

$$\delta(x)^- = max\{\delta(y)^- / Pred_{\mathcal{E}}(y) \subset Pred_{\mathcal{E}}(x)\} + 1$$
$$\delta(x)^+ = min\{\delta(y)^- / \theta(x,y)\}$$

is an interval representation of $\mathcal{E}$.

It corresponds to an ordering of the maximal antichains of $\mathcal{E}$ according to the property (iii) of theorem 1: $\delta(x)^-$ (resp. $\delta(x)^+$) is the number of the first (resp. the last) antichain containing x. But it seems to be difficult to compute this representation "on the fly" during a distributed execution. More generally there is the following result:

Theorem 5 *Let $\mathcal{E} = (\mathcal{E}, \theta)$ be an interval order. The set $([\delta(x)^-, \delta(x)^+[)_{x \in E}$ where:*

$$\delta^- \text{ satisfies } \forall x, y \in E,\ \delta(x)^- \leq \delta(y)^- \Longrightarrow Pred_{\mathcal{E}}(x) \subseteq Pred_{\mathcal{E}}(y)$$
$$\text{and } \delta(x)^+ = min\{\delta(y)^- / \theta_{im}(x,y)\}$$

is an interval representation of $\mathcal{E}$.

The proof is based on the proof of theorem 4 given in [PY79].

Proof: We have to prove that: $\forall x, y \in E,\ \theta(x,y) \Longleftrightarrow \delta(x)^+ \leq \delta(y)^-$.
$\Longrightarrow$ is obvious because of the definition of $\delta(x)^+$:
$min\{\delta(y)^- / \theta_{im}(x,y)\} = min\{\delta(y)^- / \theta(x,y)\}$.

Conversely let x and y be such that $\delta(x)^+ \leq \delta(y)^-$.
There exists z such that $\theta(x,z)$ and $\delta(x)^+ = \delta(z)^-$.
Therefore $\delta(z)^- \leq \delta(y)^-$ and $x \in Pred_{\mathcal{E}}(z) \subseteq Pred_{\mathcal{E}}(y) : \theta(x,y)$. □

For instance the function:

$$\delta(x)^- = |Pred_{\mathcal{E}}(x)|$$

satisfies the assumption of theorem 5. In fact let x and y be such that $|Pred_{\mathcal{E}}(x)| \leq |Pred_{\mathcal{E}}(y)|$. The predecessor sets of $\mathcal{E}$ are totally ordered by inclusion (see theorem 1) and therefore we necessarily have $Pred_{\mathcal{E}}(x) \subseteq Pred_{\mathcal{E}}(y)$.

[3] By convention $max(\emptyset) = 0$

Timestamping mechanism

We now assume that the causality order of the distributed computation is an interval order. This assumption is of course unrealistic, the order "$2 \oplus 2$" (theorem 1) is unavoidable during a distributed computation. But with this assumption, for each event e, the following mechanism computes the stamp $[|Pred_{\mathcal{E}}(e)|, min\{\delta(f)^- / \theta_{im}(e,f)\}[$ which codes exactly the causality order.

- If e is an internal event:
  $Pred_{\mathcal{E}}(e) = Pred_{\mathcal{E}}(e^-) \cup \{e^-\}$ and therefore
  $\delta(e)^- = \delta(e^-)^- + 1$ and $\delta(e)^+ = \delta(e^+)^-$ ($\delta(e)^+$ is known when e^+ occurs).
- If e is the sending of a message, as previously
  $\delta(e)^- = \delta(e^-)^- + 1$, but
  $\delta(e)^+ = min(\delta(e^+)^-, \delta(e^r)^-)$. We have to trace both values.
- If e is a receipt $\delta(e)^+ = \delta(e^+)^-$. To compute $\delta(e)^-$ we can have either e^- or e^e or the two immediate predecessors of e. There are two cases:
  - If $\theta(e^-, e^e)$ or $\theta(e^e, e^-)$ then $\delta(e)^- = max(\delta(e^e)^-, \delta(e^-)^-) + 1$
  - If e^- et e^e are concurrent then $\delta(e)^- = max(\delta(e^e)^-, \delta(e^-)^-) + 2$

Determining *on line* the relation existing between two events seems to be insolvable without the vector stamps. In the general case we cannot say whether we must perform
$\delta(e)^- = max(\delta(e^e)^-, \delta(e^-)^-) + 1$ or $+2$.

If the causality order of the execution is not an interval order it is clear that performing $+1$ or $+2$ preserves the message causality ordering: it orders the events on a same processor and it preserves the ordering of a sending and its receipt. By this way we produce interval extensions of the causality order. If we always perform $+1$, we compute the refinement of Lamport's stamping. If we always perform $+2$ then we observe another interval extension which is generally neither better nor worse than the previous one: it can have more or less edges (see figure 6). However there is a particular class of execution for which we are able to characterize the interval orders *on line*.

3.2.2 The RPC case

We are now considering distributed systems where the message-passing works like a Remote Procedure Call: the sender of a message is blocked until a return message is available. Each exchange is done like according to the figure 7: we merge the receipt of a message and the acknowledgment sending event. Such systems present two advantages for us. First we can really compute a stamp $\delta(e)^-$ which characterizes the interval orders and then the second stamp $\delta(e)^+$ is known on the processor where e occurs, possibly with some delay.

Actually we have seen in the preceding paragraph that if e is a receipt then we have to know whether e^- and e^e are concurrent to compute $|Pred_{\mathcal{E}}|$. But in this case there is the following result:

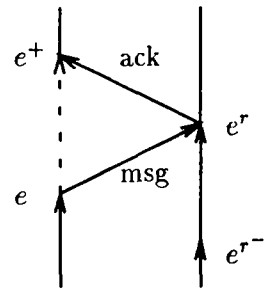

Figure 7: A synchronous execution

Proposition 3 *Let $\mathcal{E} = (E, \theta)$ be the causality order of an RPC distributed execution. If $\mathcal{E}$ is an interval order then the set of intervals $([\delta(e)^-, \delta(e)^+[)_{e \in E}$ where:*

$$\delta(e)^- = \begin{cases} \delta(e^e)^- + 1 & \text{if } e \text{ is the receipt of an acknowledgment} \\ max(\delta(e^-)^-, \delta(e^e)^-) + 2 & \text{if } e \text{ is the receipt of a message} \\ \delta(e^-) + 1 & \text{otherwise} \end{cases}$$

$$\delta(e)^+ = min\{\delta(f)^- / \theta_{im}(e, f)\}$$

is an interval representation of $\mathcal{E}$.

Proof: As before, if e is an internal event or a sending, then $|Pred_{\mathcal{E}}(e)| = |Pred_{\mathcal{E}}(e^-)| + 1$.

If e is the receipt of an acknowledgment then $\theta(e^-, e^e)$ and $Pred_{\mathcal{E}}(e) = Pred_{\mathcal{E}}(e^e) \cup \{e^e\}$. Thus $|Pred_{\mathcal{E}}(e)| = |Pred_{\mathcal{E}}(e^e)| + 1$.

If e the receipt of a message, then $|Pred_{\mathcal{E}}(e)| = max(|Pred_{\mathcal{E}}(e^-)|, |Pred_{\mathcal{E}}(e^e)|) + (1 \text{ or } 2)$ (because $\mathcal{E}$ is an interval order and the predecessors sets are included).

Thus performing
$\delta(e)^- = max(\delta(e^-)^-, \delta(e^e)^-) + 2$ may count an event twice. Therefore
$\delta(e)^- = |Pred_{\mathcal{E}}(e)| + K(e)$ where $K(e)$ corresponds to the number of sending events preceding e which have been counted twice.

Let e be the sending of a message, if e is counted once (resp. twice) when we compute $\delta(e^r)^-$ then, because of the return message, it is also counted once (resp. twice) when we compute $\delta(e^+)^-$. Therefore all the stamps of the successors of e count e either always once or always twice. This is the main difference from asynchronous systems. So we can say that $K(e)$ is an increasing function of $Pred_{\mathcal{E}}(e)$. In this way:

$$\begin{array}{lll} \forall e, f \in E, & Pred_{\mathcal{E}}(f) \subset Pred_{\mathcal{E}}(e) & \Longrightarrow K(f) \leq K(e) \\ & & \Longrightarrow \delta(f)^- < \delta(e)^- \\ i.e. & \delta(e)^- \leq \delta(f)^- & \Longrightarrow Pred_{\mathcal{E}}(e) \subseteq Pred_{\mathcal{E}}(f) \end{array}$$

because all the predecessors sets are ordered by inclusion.
Therefore δ^- satisfies the assumption of the theorem 5. □

The computation of the second stamp $\delta(e)^+$ is easy:

- If e is an internal event or the receipt of an acknowledgment then $\delta(e)^+ = \delta(e^+)^-$ and it is known with a delay.
- If e is a sending then $\theta(e^r, e^+)$ and $\delta(e)^+ = \delta(e^r)^-$. These values are known on the same site when the return message arrives: $\delta(e)^+ = \delta(e^+)^- - 1$.
- If e is the sending of an acknowledgment then $\delta(e^r)^- = \delta(e)^- + 1$ and necessarily $\delta(e)^+ = \delta(e)^- + 1$, this is known when e occurs.

The delay necessary to stamp most events is not a problem: in practice we only observe a few events and thus this delay becomes invisible. We write the corresponding algorithm:

The algorithm

```
h := 0;          (* logical clock *)
t := 0;          (* preceding value of h *)
e⁻ := "";        (* name of the preceding event *)
s := false;      (* true if the preceding event is an acknowledgment *)
At each event e ∈ E do begin
    t := h;
    case
        e is an internal event do
            h := h+1;
            if ¬s and e⁻ ∈ O then trace(e⁻, t, h);
            s := false;
        e is the sending of a message m do
            h := h+1;
            if ¬s and e⁻ ∈ O then trace(e⁻, t, h);
            s := false;
            !m(h); (* sending of m stamped by h *)
        e is the receipt of the message m stamped by h' do
            h := max(t, h')+2;
            if ¬s and e⁻ ∈ O then trace(e⁻, t, h);
            if e ∈ O then trace(e, h, h+1)
            s := true;
            !ack(h); (* sending of the acknowledgment stamped by h *)
        e is the receipt of an acknowledgment stamped by h' do
            h := h'+1;
            if e⁻ ∈ O then trace(e⁻, t, h')
            s := false;
    end
    e := e⁻;
end
```

If the message causal ordering of a distributed execution is an interval order then this algorithm allows us to observe it exactly. In the general case it produces an interval approximation which in that sense is optimum.

4 Conclusion and prospects

In this paper we have shown that the partial order theory can contribute to the elaboration of observation tools for distributed executions. Presently there is a gap between the two main stamp algorithms. Lamport's one (1978) is efficient but imprecise, Fidge and Mattern's one (1988) has a perfect precision but induces too much overhead.

We have used the class of interval orders to exhibit a new stamp technique. We have shown that it seems particularly suitable for distributed systems where message communications are performed by RPC.

In this way it seems interesting to look for other classes which can be coded by bounded size vectors. It also interesting to study the properties of the ordering induced by particular protocols. This research may lead to new stamping algorithms allowing to approach the causality order with a realistic overhead.

Acknowledgments

Special thanks are due to M. Habib, P. Baldy and C. Fioro, members of the CRIM laboratory in Montpellier, France. By long stimulating discussions, they provided most of the theoretical background on partial orders needed for this paper. We also thank B. Caillaud and JX. Rampon at IRISA for their advices and the careful reading of the paper. This work was partly funded by the French national project C^3 on concurrency.

References

[Cha91] B. Charron-Bost. Concerning the size of logical clocks in distributed systems. *Information Processing Letters*, 39(1):11–16, july 1991.

[CKS86] W.A. Cook, M. Kress, and M. Seiford. An axiomatic approach to distance on partial orderings. *R.A.I.R.O. Recherche opérationnelle/Operations Research*, 20(2):115–122, Mai 1986.

[Dil50] R.P Dilworth. A decomposition theorem for partially ordered sets. *Annals of Math.*, 51:161–165, 1950.

[Fid88] J. Fidge. Timestamps in message passing systems that preserve the partial ordering. In *Proc.* 11^{th} *Australian Computer Science Conference*, pages 55–66, 1988.

[Fis85] P.C Fishburn. *Interval orders and interval graphs.* Wiley, New York, 1985.

[Lam78] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Mat89] F. Mattern. Virtual time and global states of distributed systems. In Cosnard, Quinton, Raynal, and Robert, editors, *Proc. Int. Workshop on Parallel and Distributed Algorithms Bonas, France, Oct. 1988*, North Holland, 1989.

[Moh89] R.H. Möhring. Computationally tractable classes of ordered sets. In *Algorithms and Order*, pages 105–193, Kluwer Academic Publishers, 1989.

[PY79] C. H. Papadimitriou and M. Yannakakis. Scheduling interval-ordered task. *Siam J. Comput.*, 8(3):405–409, August 1979.

[Ram91] J.X. Rampon. Mesures de concurrence et extensions d'intervalles. Ph. D Thesis, University of Montpellier II, February 1991.

Observation d'Exécutions Réparties
Datation par Intervalles

Table des matières

1 L'observation d'exécutions réparties **18**
- 1.1 Motivations 18
- 1.2 Evénements et relation de causalité 18

2 Les algorithmes d'estampillage connus **19**
- 2.1 Estampillage par une horloge globale 19
- 2.2 L'horloge logique de Lamport 20
- 2.3 L'horloge vectorielle de Fidge et Mattern 21

3 Estampillage et ordres partiels **22**
- 3.1 Notations et rappels sur les ordres partiels 22
- 3.2 Propriétés générales d'un mécanisme d'estampillage 25
- 3.3 Estampillage vectoriel 26
- 3.4 Estampillage scalaire 28

4 Approximations intervallaires **31**
- 4.1 Amélioration de l'estampillage de Lamport 31
- 4.2 Caractérisation d'une exécution par un estampillage intervallaire 33
  - 4.2.1 Cas général 34
  - 4.2.2 Application aux exécutions distribuées 35
  - 4.2.3 Cas des exécutions synchronisées par RPC 36

5 Conclusion et perspectives **39**

1 L'observation d'exécutions réparties

1.1 Motivations

L'informatique répartie fait maintenant partie de notre quotidien mais des problèmes comme la mise au point de protocoles ou l'utilisation de machines parallèles sont très complexes et encore mal maîtrisés. Pour comprendre et analyser finement une exécution répartie il faut observer son comportement. Or, pour un système distribué, il n'existe pas d'état global qui soit observable instantanément. Si un groupe de processeurs n'a pas de base de temps commune, alors intuitivement on se sait pas ordonner totalement les événements qui se sont produits. Alors que les événements d'une exécution centralisée sont naturellement tous ordonnés, les seules relations qui existent entre les événements de deux sites distincts sont liées aux échanges de messages : un message est émis avant d'être reçu. L'ordre lié aux échanges de messages reflète la causalité des événements entre eux. C'est le seul ordre dont on est sûr qu'il a été effectivement vérifié lors de l'exécution : si deux événements n'ont aucune relation de cause entre eux et si on ne possède pas d'horloge globale pour les distinguer, on ne peut rien dire sur l'ordre dans lequel ils se sont produits.

Pour de nombreuses applications comme le diagnostic, la mesure, le monitoring, la réexécution, l'animation... ou tout simplement pour comprendre le comportement d'une exécution, il est nécessaire de connaître l'ordre dans lequel les événements ont eu lieu. Pour des raisons d'efficacité et de quantité d'information à tracer puis à stocker, il est plus intéressant d'observer et d'analyser l'ordre des événements *à la volée*, c'est à dire au cours de l'exécution. Or l'observation *à la volée* est un problème difficile : il faut dater les événements au moment où ils se produisent, mais il ne faut pas perturber l'exécution par des calculs supplémentaires ou des messages plus importants.

L'objet de cet article est donc l'étude des mécanismes d'estampillages d'exécutions réparties. Pour notre étude, nous nous situons dans un cadre formel, indépendant de tout problème d'implémentation. On considère comme système distribué, un réseau connexe de n sites identifiés : $P_1, .., P_n$. Les sites communiquent deux à deux uniquement par échange asynchrone de messages, sans perte ni duplication. On considère que le nombre de processus est constant et connu, et donc on confond la notion de processus avec celle de processeur.

1.2 Evénements et relation de causalité

Un programme réparti définit un ensemble d'actions. Une action, lorsqu'elle est exécutée, est appelée événement. Cela peut être soit une action interne à un processeur, soit l'émission d'un message vers un autre processeur, soit encore la réception d'un message. L'ensemble de ces événements est naturellement ordonné par une relation de causalité, celle mise en évidence par Lamport dans [Lam78] et appelée couramment *happened before*. On peut la définir comme la fermeture transitive des deux règles :

- les événements qui se produisent sur un même processeur sont totalement ordonnés ;
- l'envoi d'un message est antérieur à sa réception sur le processeur destinataire.

On notera θ cette relation d'ordre :

$$\theta(e,f) \Longleftrightarrow e \text{ précède causalement } f$$

Sur la figure 1, on a représenté une exécution répartie entre trois processeurs : les lignes verticales correspondent à la succession des événements d'un processeur et les lignes obliques aux échanges de messages. Sur cet exemple l'événement a précède causalement l'événement b. Cela correspond à l'existence d'un chemin reliant a à b dans le graphe dessiné.

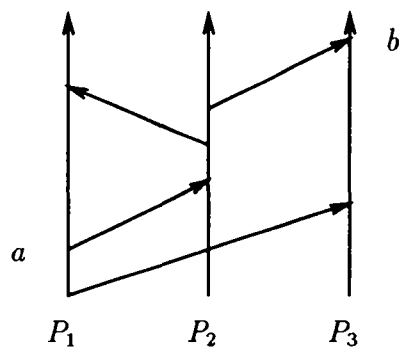

Figure 1: Représentation d'une exécution répartie

En pratique, suivant son niveau d'observation, un utilisateur n'est pas intéressé par tous les événements de l'exécution. Par exemple pour vérifier une propriété d'ordonnancement entre des événements, seuls les événements intervenant dans cette propriété seront à tracer. Mais pour calculer la relation de causalité, il faut tenir compte de tous les événements de communication. C'est pour cela que l'on distinguera l'ensemble E des événements intervenant dans le calcul de l'estampillage et O le sous-ensemble de E qui contient les événements effectivement observés. De même on note $\mathcal{E} = (E, \theta)$ l'ordre causal associé à E et $\mathcal{O} = (O, \theta)$ le sous-ordre associé à O.

2 Les algorithmes d'estampillage connus

Ce paragraphe rappelle les principales techniques de datation d'événements utilisées : datation par un temps commun indépendant de la notion de causalité et algorithmes permettant d'observer exactement ou partiellement la relation de précédence causale. Pour une bonne synthèse sur le sujet , on peut citer l'ouvrage [Ray91].

2.1 Estampillage par une horloge globale

Une méthode naturelle pour estampiller les événements d'une exécution répartie consiste à utiliser une horloge commune à tous les processeurs. Une telle horloge peut exister dans le système hôte, par exemple elle peut être délivrée localement sur chaque site par un bus. Mais dans la plupart des cas on ne dispose que d'une horloge locale sur chacun des sites. Synchroniser les horloges d'un système de processeurs ne communiquant que par échange de messages est un problème complexe. En effet les heures indiquées par deux horloges différentes

présentent *a priori* un décalage qui évolue au cours du temps. Et si on désire observer finement le comportement d'un système réparti, ces horloges ne sont pas utilisables telles quelles. Les algorithmes de synchronisation existants sont plus ou moins complexes et coûteux en fonction des hypothèses faites sur le réseau et les processeurs : la plupart rajoute des messages de resynchronisation au cours de l'exécution, la perturbant notablement.

Toutefois dans le cadre restreint des réseaux homogènes où on suppose qu'il n'y a pas de processeurs défaillants, il existe des algorithmes non intrusifs qui permettent l'utilisation non continue d'un service de temps global dont on sait, dans une certaine mesure, contrôler la précision [Jez89].

Quelque soit sa nature, physique ou reconstruite par voie logicielle, une horloge globale aura toujours une précision finie et ne permettra pas d'ordonner totalement et de façon fiable les événements. L'ordre fourni est généralement complémentaire de la relation de causalité : deux événements ordonnés par une horloge globale peuvent être, soit causalement dépendants, soit causalement indépendants. Mais si l'intervalle maximum d'incertitude du temps global est inférieur à la moitié du temps de transfert minimum d'un message alors deux événements non ordonnés par l'horloge globale et appartenant à deux sites distincts sont causalement indépendants : aucun échange de message n'a pu avoir lieu entre eux deux.

L'hypothèse sur la précision de l'horloge globale est réaliste pour quelques systèmes répartis. Dans le cas général, considérer en plus de l'ordre causal, l'ordre donné par un temps global permet d'ajouter des relations à l'ordre partiel observé. L'observation de l'exécution est donc plus précise.

2.2 L'horloge logique de Lamport

Lamport, dans l'article où il introduit la relation de précédence causale *happened before*, présente aussi un concept d'horloge logique. Pour dater les événements, chaque processeur P_i gère une horloge h_i qui varie en fonction des actions qui se produisent. Chaque événement e du site P_i peut alors être daté par une fonction δ_l : $\delta_l(e) = h_i$, au moment où e se produit. Les horloges évoluent de façon à respecter l'ordre de causalité de l'exécution :

- chaque site initialise son horloge à 0 ;
- lorsque le processus P_i exécute un événement interne ou envoie un message, son horloge est incrémentée :
  $h_i := h_i + 1$;
- chaque message émis par P_i est estampillé par la valeur de h_i au moment de l'émission ;
- quand P_i reçoit un message estampillé par t, il augmente son horloge de façon à ce qu'elle soit supérieure à l'estampille du message et à sa propre horloge :
  $h_i := max(h_i, t) + 1$.

L'ordre sur les estampilles respecte les deux règles définissant la relation de causalité : les événements d'un même site sont totalement ordonnés et une émission de message précède sa réception. Ainsi on a la propriété :

$$\boxed{\forall e, f \in E,\ \theta(e, f) \Longrightarrow \delta_l(e) < \delta_l(f)}$$

La réciproque n'est pas vraie : deux événements e et f tels que $\delta_l(e) < \delta_l(f)$ peuvent être soit causalement ordonnés soit causalement indépendants.

Cet algorithme d'estampillage, même si il ne code pas exactement la relation de causalité d'une exécution est intéressant car il est simple à mettre en œuvre : c'est le mécanisme le plus élémentaire permettant d'ordonner les événements en respectant la relation de causalité ; il y a peu de calculs supplémentaires et les messages ne sont estampillés que par un entier. D'autre part, il permet de calculer aisément un ordre total des événements : il suffit par exemple d'ordonner les événements qui ont la même date en fonction de l'identité de leur site. Cet ordre total est utilisé pour résoudre des problèmes d'algorithmique répartie : exclusion mutuelle, synchronisation...

2.3 L'horloge vectorielle de Fidge et Mattern

Fidge et Mattern [Mat89, Fid88] ont proposé un mécanisme d'horloge semblable à celui de Lamport, mais appliqué à des vecteurs d'entiers. Il permet de caractériser exactement la relation de causalité d'une exécution.

Chaque processeur gère une horloge h_i qui est un vecteur de n entiers (n, nombre total de processeurs du système observé). Si on note δ_i, le vecteur de taille n dont toutes les composantes sont nulles sauf la $i^{ème}$ égale à 1, alors les horloges évoluent suivant les règles :

- chaque site initialise son horloge au vecteur nul ;
- lorsque P_i exécute un événement interne ou envoie un message, on incrémente la $i^{ème}$ composante de son horloge :
  $h_i := h_i + \delta_i$;
- chaque message émis par P_i est estampillé par la valeur de h_i au moment de l'émission ;
- quand P_i reçoit un message estampillé par t, il compare chaque composante de son horloge avec les composantes correspondantes de l'estampille du message, et il incrémente sa $i^{ème}$ composante[1] :
  $h_i := max(h_i, t) + \delta_i$.

Cette horloge vectorielle préserve aussi la relation de causalité mais contrairement à celle de Lamport, elle permet aussi de détecter l'indépendance causale entre deux événements : elle réalise exactement un plongement de l'ordre des événements dans l'ordre des vecteurs d'entiers. Si on note δ_v la fonction de datation correspondant à cette horloge, on a la propriété :

$$\boxed{\forall e, f \in E,\ \theta(e,f) \Longleftrightarrow \delta_v(e) <_{\mathbb{N}^n} \delta_v(f)}$$

Où $<_{\mathbb{N}^n}$ est l'ordre canonique des vecteurs d'entiers de $\mathbb{N}^n$:

$$\forall x, y \in \mathbb{N}^n,\ x <_{\mathbb{N}^k} y \Longleftrightarrow \forall i \in \{1..n\},\ x[i] \leq y[i] \text{ et } \exists j \in \{1..n\},\ x[j] < y[j].$$

[1] Si on note $x[i]$ la $i^{ème}$ composante d'un vecteur x alors le maximum entre deux vecteurs est défini par : $\forall i,\ max(x,y)[i] = max(x[i], y[i])$

Les horloges vectorielles de Fidge et Mattern ont des applications dans le cadre de l'algorithmiqu répartie dès que l'on a besoin de savoir si des événements sont indépendants ou non, comme par exemple pour déboguer des programmes répartis ou pour faire des mesures de concurrence [Cha89, Ram91]. Toutefois l'utilisation de ces horloges n'est réaliste que pour des systèmes où le nombre de processeurs est petit. Chaque message devant véhiculer une estampille de taille n, la perturbation induite devient rapidement trop importante. Si on a gagné en précision par rapport à l'horloge de Lamport, on a perdu énormément en complexité : on est passé d'une estampille de taille constante, à une estampille de taille proportionnelle au nombre de sites du système observé. La taille que l'on considère ici n'est ni l'ordre de grandeur que peut atteindre l'estampille : celle de Lamport peut croître indéfiniment, ni la taille du vecteur servant d'estampille : n entiers peuvent être codés sur un nombre d'octets plus petit. La taille d'une estampille est vue ici comme le nombre de paramètres qui varient indépendamment les uns des autres. D'ailleurs, comme cela est montré dans [Cha91], si on cherche un plongement de la relation de causalité dans l'ordre des vecteurs d'entiers, la taille n est inévitable. C'est pourquoi nous sommes intéressé par des estampillages approchant la relation de causalité qui sont de taille constante.

Dans le paragraphe suivant nous interprétons en terme d'ordres partiels ces deux horloges et nous voyons comment la théorie des ordres peut nous aider à trouver d'autres mécanismes d'estampillage. La distance qui sépare ces deux algorithmes est suffisamment grande, tant en complexité qu'en précision pour que l'on puisse penser qu'il existe des algorithmes intermédiaires qui, au prix d'hypothèses sur l'exécution observée, donnent des résultats satisfaisants.

3 Estampillage et ordres partiels

3.1 Notations et rappels sur les ordres partiels

Pour représenter une exécution répartie nous dessinerons le graphe de couverture orienté représentant l'ordre de causalité. C'est le graphe où les sommets sont les éléments de l'ordre et les arcs sont les relations de comparabilité. Pour obtenir une représentation compacte, on supprime les arcs de transitivité et on oriente implicitement les arêtes du bas vers le haut. Sur la figure 2 on a un exemple d'exécution répartie. On a représenté tous les événements servant au calcul de la relation de causalité et on a marqué par des rond noirs les événements effectivement observés.

Le passage à la représentation dessinée sur la figure 3 se fait en éliminant les sommets qui ne sont pas observés mais sans oublier les relations qu'ils entraînaient entre les autres événements. Cet exemple d'exécution sera repris tout au long de l'article pour présenter et comparer les différents algorithmes d'estampillage.

On définit la **taille** $|\mathcal{E}|$ d'un ordre $\mathcal{E} = (E, \theta)$ comme étant le nombre total de relations de θ, ou encore le nombre d'arcs de la fermeture transitive du graphe représentant $\mathcal{E}$. Par exemple la taille du graphe représenté figure 3 est 13.

Une **chaîne** est un sous-ensemble de E totalement ordonné. Inversement, une **antichaîne** est un sous-ensemble de E dans lequel tous les éléments sont incomparables.

La **largeur** d'un ordre est égale à la taille d'une de ces plus grandes antichaînes

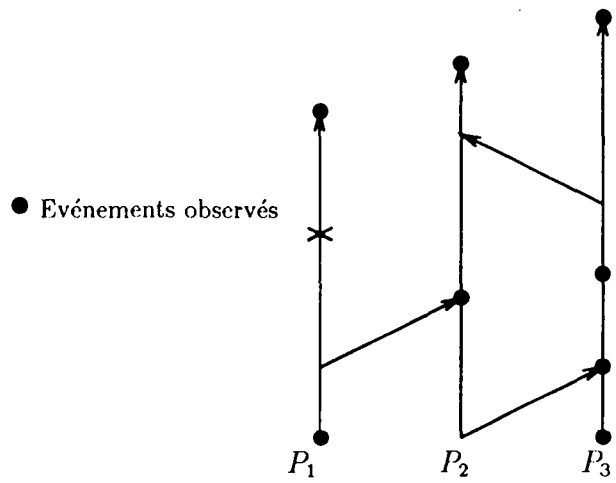

Figure 2: Une exécution répartie

y **couvre** x pour l'ordre $\mathcal{E} = (E, \theta)$ et on note $\theta_{im}(x, y)$ quand y succède immédiatement à x, c'est à dire :

$$\theta_{im}(x, y) \Longleftrightarrow \theta(x, y) \text{ et } \nexists z \in E,\ \theta(x, z) \text{ et } \theta(z, x).$$

On définit les ensembles prédécesseurs et prédécesseurs immédiats :

$$Pred_{\mathcal{E}}(x) = \{y \in E / \theta(y, x)\} \text{ et } Pred_{im\mathcal{E}}(x) = \{y \in E / \theta_{im}(y, x)\}$$

Lors des énoncés des mécanismes d'estampillage, on se servira des notations suivantes : si e est un événement, on appelle e^- (resp. e^+) l'événement précédant (resp. suivant) immédiatement e sur le même processeur. Si e est une émission de message, on appelle e^r l'événement de réception de ce même message sur le site destinataire. Et enfin si e est une réception, on note e^e l'événement qui correspond à son émission. Ces notations sont résumées sur la figure 4.

Dans le paragraphe 4 on s'intéresse à la classe des **ordres d'intervalles**. Ce sont des ordres qui modélisent la structure séquentielle des intervalles de la droite réelle. Ils trouvent donc leur intérêt dans les applications utilisant le concept de temps et ont, de ce fait, été largement étudiés ces dernières années : on peut par exemple citer l'ouvrage de référence sur le sujet : [Fis85] et les travaux de [Ram91].

Définition 1 *Un ordre (E, θ) est un ordre d'intervalles si et seulement si on peut associer à tout élément x de E un intervalle I_x de la droite réelle tel que deux éléments sont ordonnés si l'intervalle de l'un est à la gauche de l'intervalle de l'autre :*

$$\forall x, y \in E, \theta(x, y) \Longleftrightarrow I_x \textit{ est à la gauche de } I_y$$

Il existe de nombreuses caractérisations des ordres d'intervalles. Celles qui nous servirons dans la suite sont résumées dans le théorème suivant [Moh89].

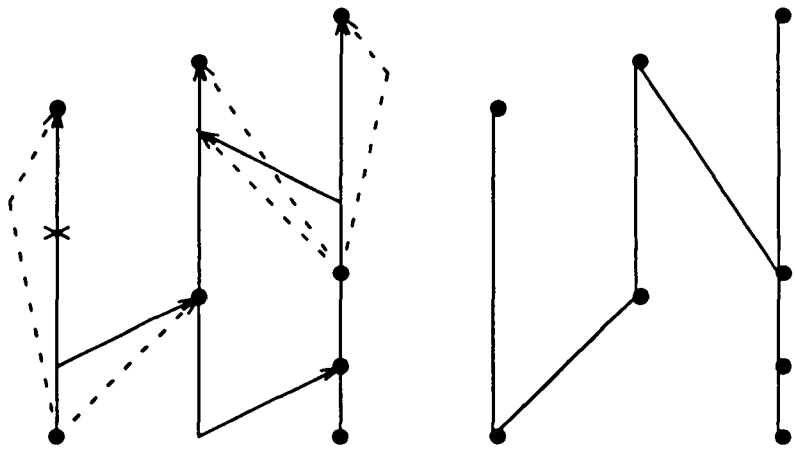

Figure 3: Construction de la représentation associée à l'exécution de la figure précedente

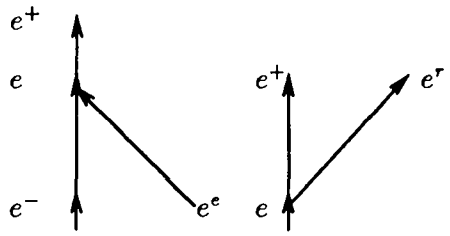

Figure 4: Notations

Théorème 1 *Pour un ordre $\mathcal{E} = (E, \theta)$ les conditions suivantes sont équivalentes :*

(i) *$\mathcal{E}$ est un ordre d'intervalles.*

(ii) *$\mathcal{E}$ ne contient pas de sous-ordre isomorphe à l'ordre "$2 \oplus 2$" (figure 5).*

(iii) *Les antichaînes maximales de $\mathcal{E}$ peuvent être linéairement ordonnées de telle sorte que pour tout x de E les antichaînes maximales contenant x apparaissent consécutivement.*

(iv) *Les ensembles prédécesseurs $Pred_{\mathcal{E}}(x)$ sont linéairement ordonnés par inclusion.*

Un lecteur non familier des ordres partiels peut se convaincre de ces résultats en considérant la structure intervallaire des ordres. Par exemple sur la figure 6, on voit qu'il n'est pas possible de représenter l'ordre "$2 \oplus 2$" avec des intervalles, on rajoute nécessairement des relations entre des sommets incomparables.

Pour dater des exécutions réparties, la classe des ordres d'intervalles semble particulièrement intéressante. Chaque élément de l'ordre peut être représenté par un intervalle de la droite réelle donc par un couple de réels. Il existe donc pour les ordres d'intervalles, des codages de taille deux.

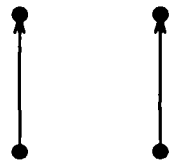

Figure 5: L'ordre partiel "$2 \oplus 2$"

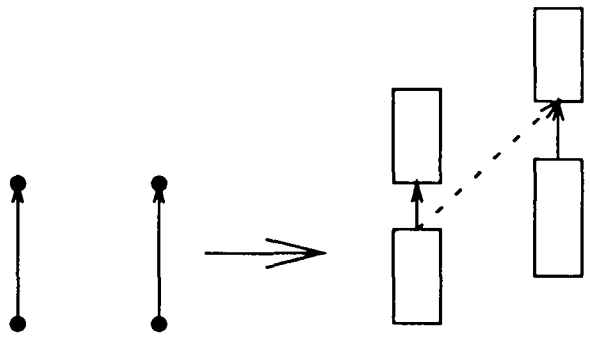

Figure 6: Tentative de représentation de "$2 \oplus 2$" par des intervalles

Enfin on appelle **extension** (E, θ') d'un ordre (E, θ), tout ordre qui prolonge θ :

$$\forall x, y \in E,\ \theta(x, y) \Longrightarrow \theta'(x, y).$$

Par exemple, l'ordre produit par l'horloge logique de Lamport est une extension de l'ordre de causalité. Une **extension linéaire** est une extension qui ordonne totalement E et une **extension intervallaire** est une extension qui est un ordre d'intervalles.

3.2 Propriétés générales d'un mécanisme d'estampillage

Une technique d'estampillage consiste à associer à tout événement x, une date $\delta(x)$. Comme nous l'avons vu précédemment, cela peut-être un réel, un entier, un ensemble d'entiers... En définissant une relation d'ordre sur l'ensemble des estampilles, on définit aussi une relation sur l'ensemble des événements observés.

Coder exactement la relation de causalité d'une exécution est une opération coûteuse : des calculs supplémentaires ou des messages plus importants ralentissent l'exécution et peuvent modifier le comportement de l'algorithme. Un mécanisme d'estampillage doit donc au minimum préserver la relation de causalité. Cela correspond à l'idée que l'algorithme peut ordonner des événements qui sont concurrents, mais que si deux événements sont causalement dépendants, ils ne doivent pas être observés *à l'envers*. Il ne faut pas donner à un utilisateur une information qui est intuitivement fausse.

Nous comparerons les algorithmes d'estampillage d'une part en évaluant la perturbation qu'ils induisent sur l'exécution : taille des estampilles rajoutées aux messages, calculs supplémentaires ; et d'autre part en calculant la taille des ordres qu'ils produisent sur une même exécution. Ce critère permet de comparer différentes extensions d'un même ordre, il a été introduit dans [CKS86].

Un mécanisme estampillage permet d'ordonner instantanément des événements. Le calcul des estampilles doit donc pouvoir se faire *à la volée*, c'est à dire au moment de l'exécution d'une action. Il doit donc dépendre que des événements qui l'ont précédés :

$$\delta(x) = f(Pred_{\mathcal{E}}(x)).$$

Toutefois on envisage dans la suite des algorithmes où l'estampille de certains événements dépend de leur futur, mais il faut veiller à ce que ce futur reste proche. Dans ce cas on dira que l'algorithme n'est plus incrémental mais seulement *pseudo-incrémental.*

Enfin pour minimiser la perturbation induite, le calcul d'une estampille doit être fait à partir des informations locales d'un site : il ne doit pas ajouter d'échange de messages.

3.3 Estampillage vectoriel

Nous allons montrer que l'estampillage d'une exécution répartie par un vecteur d'entiers est l'illustration d'une technique générale de codage des ordres partiels.

Le **théorème de Dilworth** [Dil50] nous dit qu'un ordre de largeur k peut se décomposer en k chaînes disjointes.

$$E = \bigcup_{1 \leq i \leq k} L_i, L_i \text{chaîne}$$

Une telle décomposition permet de construire un codage de l'ordre : on associe à chaque élément x un vecteur dont la $i^{ème}$ composante est le nombre de prédécesseurs de x qui appartiennent à la chaîne L_i. Le théorème suivant est probablement un cas particulier d'une propriété plus générale mais l'écriture de sa démonstration permet de mieux comprendre la structure du codage.

Théorème 2 *Soit $\mathcal{E} = (E, \theta)$ un ensemble partiellement ordonné de largeur k et $(L_i)_{1 \leq i \leq k}$ une décomposition en chaînes de E. L'application :*

$$\begin{aligned} \phi \ : \ & E \longrightarrow \mathbb{N}^k \\ & x \longmapsto (|(Pred_{\mathcal{E}}(x) \cup \{x\}) \cap L_i|)_{1 \leq i \leq k} \end{aligned}$$

est un plongement dans l'ordre $<_{\mathbb{N}^n}$ des vecteurs d'entiers (définition au paragraphe 2.3) c'est à dire satisfait :

$$\forall x, y \in E, \ \theta(x,y) \Longleftrightarrow \phi(x) <_{\mathbb{N}^k} \phi(y).$$

Preuve On considère deux éléments de E, x et y tels que $\theta(x,y)$, c'est à dire $x \in Pred_{\mathcal{E}}(y)$.
On a donc en particulier
$Pred_{\mathcal{E}}(x) \cup \{x\} \subset Pred_{\mathcal{E}}(y) \cup \{y\}$
L'inclusion reste vraie quand on fait l'intersection avec les chaînes L_i
$(Pred_{\mathcal{E}}(x) \cup \{x\}) \cap L_i \subseteq (Pred_{\mathcal{E}}(y) \cup \{y\}) \cap L_i$
et ainsi en prenant le cardinal de ces ensembles on a bien
$\forall i \in \{1..k\}, \ \phi(x)_i \leq \phi(y)_i$.
y ne peut pas appartenir à $Pred_{\mathcal{E}}(x)$ car on ne peut pas avoir $\theta(y,x)$ et il existe un indice j tel que $y \in L_j$, on a ainsi l'inclusion stricte :
$(Pred_{\mathcal{E}}(x) \cup \{x\}) \cap L_j \subset (Pred_{\mathcal{E}}(y) \cup \{y\}) \cap L_j$
et donc : $\exists j \in \{1..k\}, \ \phi(x)_j < \phi(y)_j$. On a ainsi montré l'implication $\Longrightarrow$.

Réciproquement, soit x et y tels que : $\phi(x) <_{\mathbb{N}^k} \phi(y)$
c'est à dire en particulier
$\forall i \in \{1..k\}, \ |(Pred_{\mathcal{E}}(x) \cup \{x\}) \cap L_i| \leq |(Pred_{\mathcal{E}}(y) \cup \{y\}) \cap L_i|$.
Les éléments de chaque L_i sont totalement ordonnés, on a donc nécessairement
$(Pred_{\mathcal{E}}(x) \cup \{x\}) \cap L_i \subseteq (Pred_{\mathcal{E}}(y) \cup \{y\}) \cap L_i$.
Les ensembles L_i réalisant une partition de E on a
$Pred_{\mathcal{E}}(x) \cup \{x\} \subset Pred_{\mathcal{E}}(y) \cup \{y\}$,
et par conséquent $x \in Pred_{\mathcal{E}}(y)$, c'est à dire $\theta(x,y)$. On a montré l'implication réciproque $\Longleftarrow$.

□

Donc si on connaît une décomposition en chaînes d'un ordre partiel, on peut en déduire un codage de l'ordre. Or pour l'observation des exécutions réparties on a une décomposition en chaînes triviale : chaque chaîne L_i est composée des événements ayant eut lieu sur le processeur P_i. La taille de cette décomposition n'est pas nécessairement égale à la largeur de l'ordre, donc n'est pas optimale, mais elle a le mérite d'être immédiate. Si il y a n processeurs dans le système :

$$E = \biguplus_{1 \leq i \leq n} L_i, \ L_i \text{ contient les événements se produisant sur } P_i.$$

Le plongement défini dans le théorème 2 peut être calculé incrémentalement au cours d'une exécution. Pour cela chaque processeur maintient à jour un vecteur de taille n, $\delta(e) \in \mathbb{N}^n$ et le fait évoluer par les règles suivantes.

- Lorsque le premier événement du site P_i se produit, sauf si c'est une réception de message, son ensemble de prédécesseurs est vide et donc le vecteur du site est initialisé à δ_i :

$$\delta(e) = \delta_i$$

(rappel : δ_i est le vecteur dont toute les coordonnées sont nulles sauf la $i^{ème}$ égale à 1).

- Si l'événement e du site n°i est un événement interne ou une émission de message alors $Pred(e) = Pred(e^-) \cup \{e^-\}$ et ainsi

$$\delta(e) = \delta(e^-) + \delta_i.$$

- si l'événement e est une réception de message alors e peut avoir deux prédécesseurs immédiats e^- et e^e, un seul si il existe déjà une relation entre eux. D'où exprimé avec les ensembles prédécesseurs :
$Pred_{\mathcal{E}}(e) = Pred_{\mathcal{E}}(e^-) \cup Pred_{\mathcal{E}}(e^e) \cup \{e^-, e^e\}$.
Les ensembles L_i étant totalement ordonnés, on a les relations d'inclusion
$\forall i \in \{1..n\}, Pred_{\mathcal{E}}(e^e) \cap L_i \subset$ (ou $\supset$) $Pred_{\mathcal{E}}(e^-) \cap L_i$.
On peut alors calculer le cardinal de $Pred_{\mathcal{E}}(e)$
$|Pred_{\mathcal{E}}(e) \cap L_j| = max(|(Pred_{\mathcal{E}}(e^e) \cup \{e^e\}) \cap L_j|, |(Pred_{\mathcal{E}}(e^-) \cup \{e^-\}) \cap L_j|)$
et ainsi

$$\delta(e) = max(\delta(e^-), \delta(e^e)) + \delta_i$$

Pour calculer la nouvelle valeur de l'horloge, il faut donc connaître la valeur de celle du site émetteur au moment de l'envoi du message.

On reconnaît le mécanisme des horloges logiques proposées par Fidge et Mattern. On montre ainsi qu'elles codent exactement la relation de causalité d'une exécution. Plus précisément, l'estampille d'un événement caractérise son passé : la $i^{ème}$ composante de l'estampille est le nombre d'événements qui l'ont causalement précédés sur le site P_i. Pour un exemple d'application voir la figure 7.

3.4 Estampillage scalaire

Le codage du paragraphe précédent est basé sur une décomposition en chaîne des ordres. Nous allons maintenant nous intéresser à une décomposition en antichaînes. Pour cela on définit tout d'abord la notion de rang.

Définition 2 *Soit $\mathcal{E} = (E, \theta)$ un ensemble partiellement ordonné. Le rang d'un élément x, $\rho(x)$, est la longueur maximale d'une chaîne se terminant en x.*

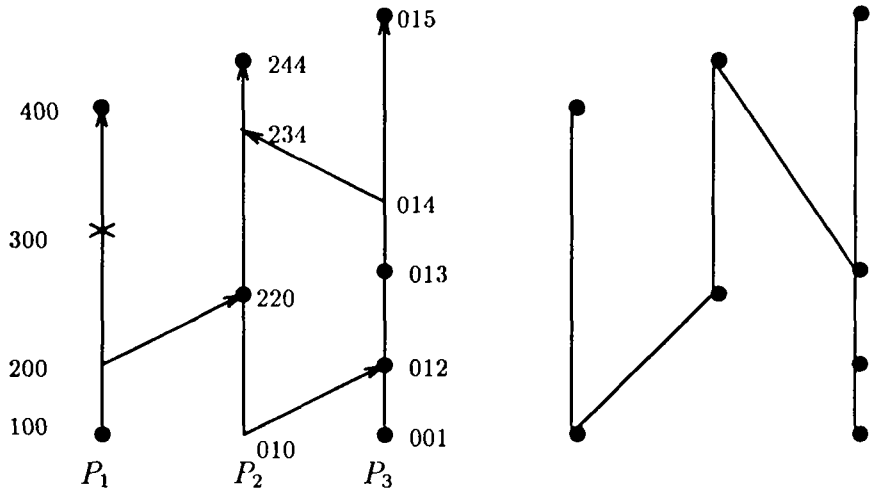

Figure 7: Estampillage de Mattern et ordre observé : $|\mathcal{E}| = |\mathcal{O}| = 13$

Les éléments d'un ensemble ordonné peuvent être partitionnés en fonction de la valeur de leur rang.

Définition 3 *Soit $\mathcal{E} = (E, \theta)$ un ensemble partiellement ordonné fini. La décomposition :*

$$E = \biguplus_{0 \leq i \leq h} R_i \text{ où } \forall i \geq 0 \; R_i = \{x \in E / \rho(x) = i\}$$

est appelée la décomposition par rang de $\mathcal{E}$.

Deux éléments x et y ayant même rang sont concurrents. En effet, si on avait par exemple $\theta(x, y)$, on pourrait construire en rajoutant x à une chaîne maximale se terminant en x, une chaîne se terminant en y de longueur $\rho(x) + 1$. D'où contradiction. Chaque sous-ensemble R_i est donc une antichaîne. Ces antichaînes ne sont pas maximales. Par contre le nombre d'antichaînes de la décomposition est minimal.

Le théorème 3 donne un algorithme de construction de la décomposition. Il correspond à un parcours de graphe classique : la descente en couche. La première couche est composée des éléments minimaux. On les enlève et on prend comme deuxième couche les nouveaux minimaux, on les enlève et ainsi de suite...

Théorème 3 *Soit $\mathcal{E} = (E, \theta)$ un ensemble partiellement ordonné fini. La décomposition :*

$$E = \biguplus_{0 \leq i \leq h} A_i \text{ où } \begin{array}{l} A_0 = Min(E, \theta) = \{x \in E / \nexists y \in E, \; \theta(y, x)\} \\ \text{et } \forall i > 0 \; A_i = Min(E - \bigcup_{j<i} A_j, \theta) \end{array}$$

est la décomposition par rang de $\mathcal{E}$, c'est à dire $\forall i \in \{1..h\}$, $R_i = A_i$.

Preuve Soit $i \geq 0$ et $x \in A_i$. D'après la définition des ensembles A_j, on peut construire une chaîne qui se termine en x et qui contient au moins un élément dans chacun des ensembles A_j, $j < i$. Du fait que c'est une chaîne, elle ne peut pas contenir plusieurs éléments appartenant au même A_j (les A_j sont composés d'éléments incomparables). Ainsi la longueur maximale d'une chaîne se terminant en x est i. On a montré : $\rho(x) = i$.

□

On peut exprimer le rang d'un élément en fonction du rang de ces prédécesseurs : pour un élément x de A_i ($i > 0$), par définition des ensembles A_j au moins un prédécesseur immédiat de x appartient à A_{i-1}. Donc :

$$\rho(x) = max\{\rho(y),\ \theta_{im}(y, x)\} + 1$$

Si $\mathcal{E} = (E, \theta)$ est la relation de causalité d'une exécution répartie, cette expression nous fournit un mécanisme de calcul incrémental de la fonction de rang.

- Si e est le premier événement se produisant sur un site, sauf une réception de message, alors e est minimal pour l'ordre de causalité, c'est à dire $e \in A_0$ et donc :

$$\rho(e) = 0.$$

- Si e est un événement interne ou l'émission d'un message, alors e n'a qu'un seul prédécesseur immédiat qui est l'événement le précédant sur le même site : e^-. On a donc :

$$\rho(e) = \rho(e^-) + 1.$$

- Si e est la réception d'un message, E peut avoir au plus deux prédécesseurs immédiats : $Pred_{im\mathcal{E}}(e) \subseteq \{e^-, e^e\}$. D'où :

$$\rho(e) = max(\rho(e^-), \rho(e^e)) + 1.$$

Pour calculer $\rho(e)$ il faut connaître le rang de l'événement d'émission.

On reconnaît les règles d'évolution de l'horloge de Lamport.

Les horloges vectorielles de Fidge et Mattern sont basées sur une décomposition en chaînes de la relation de causalité et on vient de voir que l'horloge de Lamport correspond à une décomposition en antichaînes.

La figure 8 donne une idée de la structure de l'extension produite par cet estampillage. Cette structure est exactement celle des ordres faibles : un **ordre faible** est un ordre total dans lequel certains éléments ont été remplacés par des antichaînes. Le diagramme d'un tel ordre correspond à une succession d'antichaînes totalement connectées entre elles. On peut donc énoncer la proposition :

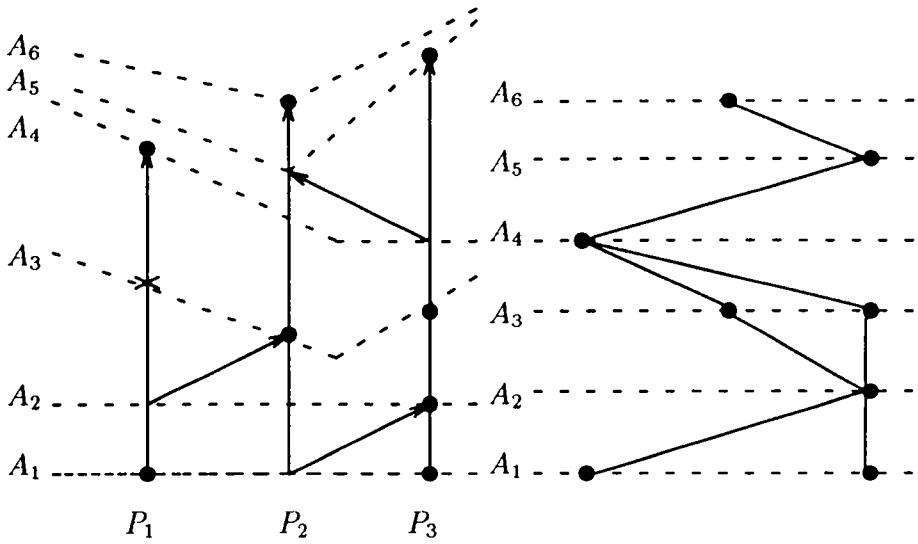

Figure 8: Estampillage de Lamport et ordre observé :$|\mathcal{O}| = 26$

Proposition 1 *L'algorithme d'estampillage proposé par Lamport dans [Lam78] calcule le rang de chaque événement pour l'ordre de causalité et l'ordre produit est un ordre faible.*

Nous avons montré que les deux algorithmes d'estampillage classiques reposent sur des propriétés simples et connues de la théorie de ordres partiels. Nous allons donc voir maintenant, comment en s'inspirant d'une classe d'ordres particulière, on peut apporter des améliorations aux algorithmes existants et trouver des classes d'exécutions caractérisables par un estampillage de taille bornée.

4 Approximations intervallaires

4.1 Amélioration de l'estampillage de Lamport

L'ordre donné par les estampilles de Lamport peut être considéré comme un ordre d'intervalles. En effet, on peut associer à chaque événement e l'intervalle réel $[\rho(e), \rho(e) + 1[$ qui préserve la relation d'ordre. Les intervalles associés aux événements étant tous de même longueur, l'ordre défini par les estampilles de Lamport est un **semi-ordre** (un semi-ordre est un ordre d'intervalles pour lequel il existe une représentation avec des intervalles de longueur tous identique). Le schéma de la figure 9 permet de visualiser sa structure intervallaire.

Cette remarque nous donne une idée pour améliorer l'approximation donnée par Lamport. On associe à chaque événement un intervalle $[\delta(e)^-, \delta(e)^+[$ où $\delta(e)^- = \rho(e)$, l'estampille de Lamport, et $\delta(e)^+$ une valeur qui est au moins supérieure à $\rho(e)+1$. Cette valeur est choisie de façon à respecter la relation de causalité. On allonge ainsi la taille de certains intervalles, cela

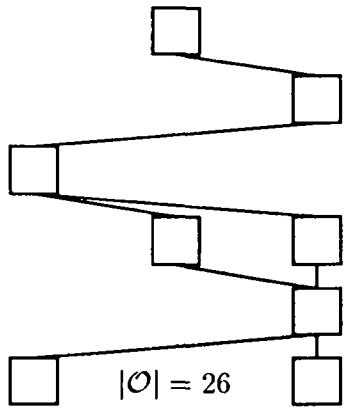

Figure 9: Représentation intervallaire de l'estampillage de Lamport

permet d'éliminer des relations entre des événements. On se rapproche ainsi de la relation de causalité. La figure 10 montre un cas où on peut agrandir un intervalle : l'horloge de Lamport ordonne les deux événements e^- et e^e qui sont causalement indépendants. Or le premier successeur de e^- est e. On peut donc prolonger l'intervalle de e^- jusqu'au début de celui de e, on respecte la relation de causalité tout en éliminant la dépendance entre e^- et e^e.

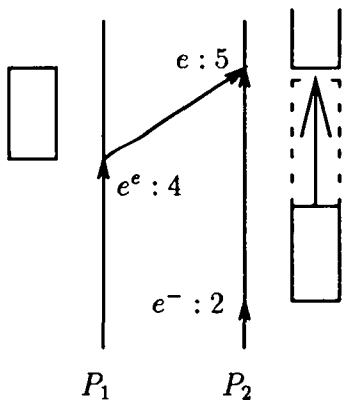

Figure 10: Principe de l'amélioration de l'estampillage de Lamport

Plus formellement, on a la proposition :

Proposition 2 *L'ensemble d'intervalles* $([\delta(e)^-, \delta(e)^+[)_{e \in E}$ *où*

$$\delta(e)^- = \rho(e), \textit{ l'estampille de Lamport}$$
$$\delta(e)^+ = min\{\delta(f)^- / \theta_{im}(e, f)\}$$

est une approximation intervallaire de la relation de causalité.

Sur la figure 11 on voit un exemple d'application de cet estampillage : on a supprimé quatre dépendances par rapport à l'estampillage de Lamport. L'ordre obtenu est une extension intervallaire de la relation de causalité qui en terme d'inclusion d'ordres se place entre la relation de causalité et l'ordre donné par l'horloge de Lamport.

Toutefois, cette technique d'estampillage n'est plus incrémentale. La borne supérieure de l'intervalle dépend non pas du passé des événements, mais de leur futur proche. Pour connaître l'estampille complète d'un événement, il faut attendre que ces successeurs immédiats se soient produits. Dans la pratique, ce n'est pas problème : il y a *a priori* peu d'événements à tracer.

- Les événements internes et les réceptions de message n'ont qu'un seul successeur immédiat : l'événement suivant sur le même site : il suffit d'attendre qu'il se produise pour tracer l'estampille complète du précédent.

- Une émission de message e peut avoir deux successeurs immédiats : l'événement suivant sur le même site, e^+ et l'événement de réception sur le site destinataire, e^r. Pour calculer $\delta(e)^+ = min(\delta(e^+)^-, \delta(e^r)^-)$ on trace ces deux valeurs potentielles : $\delta(e^+)^-$ et $\delta(e^r)^-$ et on prend la plus petite des deux.

Cela nécessite donc un traitement sur la trace relevée, mais c'est un traitement qui peut être fait parallèlement à l'exécution et pas nécessairement à sa fin.

Comme on le voit dans la suite, dans le cas d'un système où les émissions de message se font avec un accusé de réception bloquant, ce problème est résolu : on connaît dans tous les cas l'estampille complète d'un événement lorsque que son successeur sur le même site a lieu.

La perturbation liée à l'observation est la même que celle induite par les horloges de Lamport : on ne rajoute pas de message supplémentaire et les estampilles véhiculée par les messages sont de même taille : un entier.

4.2 Caractérisation d'une exécution par un estampillage intervallaire

Dans le paragraphe précédent, on a mis en évidence deux extensions intervallaires de la relation de causalité : l'estampillage de Lamport et son amélioration. Nous cherchons maintenant à trouver un algorithme de codage de la relation de causalité qui caractérise exactement les exécutions dont l'ordre de causalité est un ordre d'intervalles.

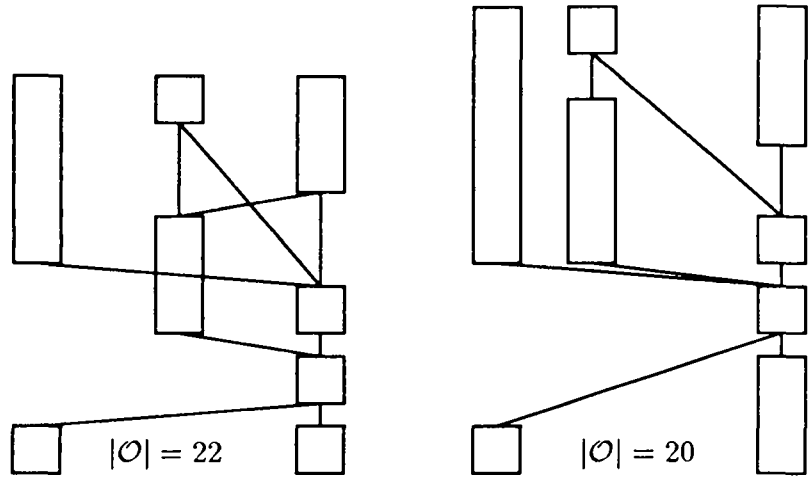

Figure 11: Représentation intervallaire de amélioration de l'estampillage de Lamport et de l'estampillage "max+2"

4.2.1 Cas général

Caractérisation des ordres d'intervalles

Le théorème suivant est le fondement de l'algorithme de reconnaissance d'ordres d'intervalles publié par Papadimitriou et Yannakakis dans [PY79]. Il permet de calculer une représentation intervallaire des ordres d'intervalles.

Théorème 4 *Soit $\mathcal{E} = (\mathcal{E}, \theta)$ un ordre d'intervalles.*
L'ensemble $([\delta(x)^-, \delta(x)^+[)_{x \in E}$ où :

$$\delta(x)^- = max\{\delta(y)^- / Pred_{\mathcal{E}}(y) \subset Pred_{\mathcal{E}}(x)\} + 1$$
$$\delta(x)^+ = min\{\delta(y)^- / \theta(x,y)\}$$

est une représentation intervallaire de $\mathcal{E}$.

Par convention on considère que $max(\emptyset) = 0$.

Les intervalles de cette représentation correspondent à un ordonnancement des antichaînes maximales de $\mathcal{E}$ suivant la caractérisation (iii) du théorème 1 : $\delta(x)^-$ (resp. $\delta(x)^+$) est le numéro de la première (resp. la dernière) antichaîne à laquelle x appartient. Il semble toutefois difficile de calculer *à la volée* cette représentation durant une exécution répartie. Nous démontrons donc le résultat plus général :

Théorème 5 *Soit $\mathcal{E} = (\mathcal{E}, \theta)$ un ordre d'intervalles. L'ensemble $([\delta(x)^-, \delta(x)^+[)_{x \in E}$ où :*

$$\delta^- \text{ satisfait } \forall x, y \in E,\ \delta(x)^- \leq \delta(y)^- \Longrightarrow Pred_{\mathcal{E}}(x) \subseteq Pred_{\mathcal{E}}(y)$$
$$\text{et } \delta(x)^+ = min\{\delta(y)^- / \theta_{im}(x, y)\}$$

est une représentation intervallaire de $\mathcal{E}$.

Preuve On doit montrer que : $\forall x, y \in E,\ \theta(x, y) \Longleftrightarrow \delta(x)^+ \leq \delta(y)^-$.
L'implication $\Longrightarrow$ est immédiate grâce à la définition de $\delta(x)^+$:
$min\{\delta(y)^- / \theta_{im}(x, y)\} = min\{\delta(y)^- / \theta(x, y)\}$.
Réciproquement, on considère x et y tels que : $\delta(x)^+ \leq \delta(y)^-$. Par définition de $\delta(x)^+$ il existe z tel que $\theta(x, z)$ et $\delta(x)^+ = \delta(z)^-$. On donc $\delta(z)^- \leq \delta(y)^-$, ce qui entraîne : $x \in Pred_{\mathcal{E}}(z) \subseteq Pred_{\mathcal{E}}(y) : \theta(x, y)$. L'implication $\Longleftarrow$ est démontrée.

□

Par exemple la fonction :

$$\delta(x)^- = |Pred_{\mathcal{E}}(x)|$$

satisfait l'hypothèse du théorème 5. En effet si on prend x et y tels que $|Pred_{\mathcal{E}}(x)| \leq |Pred_{\mathcal{E}}(y)|$, les ensembles prédécesseurs d'un ordre d'intervalles étant totalement ordonnés par l'inclusion, on a nécessairement $Pred_{\mathcal{E}}(x) \subseteq Pred_{\mathcal{E}}(y)$.

4.2.2 Application aux exécutions distribuées

On considère maintenant que l'on observe une exécution distribuée dont on suppose par avance que l'ordre causal est un ordre d'intervalles. C'est une hypothèse qui est bien sûre irréaliste, il suffit de considérer la caractérisation (ii) du théorème 1 : l'ordre "$2 \oplus 2$" (figure 5) est inévitable. Mais sous cette hypothèse et avec le résultat précédent, on sait que l'estampille $[|Pred_{\mathcal{E}}(e)|, min\{\delta(f)^- / \theta_{im}(e, f)\}[$ code exactement la relation de causalité. Le calcul est fait pseudo-incrémentalement par le mécanisme suivant.

- Si e est un événement interne : $Pred_{im\mathcal{E}}(e) = \{e^-\}$ et ainsi
  $\delta(e)^- = \delta(e^-)^- + 1$ et $\delta(e)^+ = \delta(e^+)^-$.
  $\delta(e)^+$ n'est donc pas connu lorsque e a lieu, mais lors de l'événement suivant.

- Si e est une émission de message comme précédemment $\delta(e)^- = \delta(e^-)^- + 1$, mais $\delta(e)^+ = min(\delta(e^+)^-, \delta(e^r)^-)$.
  Pour calculer $\delta(e)^+$, il faut tracer $\delta(e^+)^-$ et $\delta(e^r)^-$ et prendre la plus petite des deux valeurs.

- Si e est une réception de message on a comme pour un événement interne : $\delta(e)^+ = \delta(e^+)^-$.
  Pour calculer $\delta(e)^-$, on peut avoir soit e^-, soit e^e, soit les deux, prédécesseurs immédiats de e. Il y a donc deux cas :

- Si $\theta(e^-, e^e)$ ou $\theta(e^e, e^-)$ alors $\delta(e)^- = \delta(e^e)^- + 1$.
- Si e^- et e^e sont concurrents alors $\delta(e)^- = max(\delta(e^e)^-, \delta(e^-)^-) + 2$

Déterminer *à la volée* la relation qui existe entre deux événements semble rester un problème insoluble sans les estampilles vectorielles. Dans le cas général, on ne peut pas trancher pour savoir si il faut faire $\delta(e)^- = max(\delta(e^e)^-, \delta(e^-)^-) + 1$ ou $+2$.

Si la relation de causalité de l'exécution n'est pas un ordre d'intervalles, ajouter 1 ou 2 préserve la causalité : cela ordonne les événements qui ont lieu sur un même processeur et cela respecte l'ordre entre l'émission et la réception d'un message. On produit ainsi des extensions intervallaires. Si l'on choisit arbitrairement de faire toujours $+1$, on retrouve le calcul de l'estampille de Lamport et son extension intervallaire améliorée. Si on choisit de faire toujours $+2$, on observe une autre extension intervallaire de l'ordre qui n'est, dans le cas général, ni meilleure, ni moins bonne que celle de Lamport améliorée : elle peut comprendre des arcs en plus et des arcs en moins. Sur les figures 9 et 11 on peut comparer sur notre exemple d'exécution les approximations intervallaires données par l'estampillage de Lamport seul, par l'estampillage de Lamport où on a allongé des intervalles, et par notre estampillage "max + 2".

Toutefois, il existe un cadre particulier d'exécutions où l'on sait caractériser les ordres d'intervalles *à la volée*. C'est le cas des exécutions où les communication se font par *Remote Procedure Call*.

4.2.3 Cas des exécutions synchronisées par RPC

On considère maintenant les exécutions où les échanges de messages se font avec accusé de réception bloquant. C'est une mise en œuvre du mode de communication par *Remote Procedure Call* ou RPC. Chaque échange de message est effectué suivant le schéma de la figure 12. Le transfert du message du message a lieu entre e et e^r. On confond l'événement de réception e^r avec l'événement d'émission de l'accusé de réception. e^+ est la réception de l'accusé de réception. Entre e et e^+ aucun événement ne peut se produire.

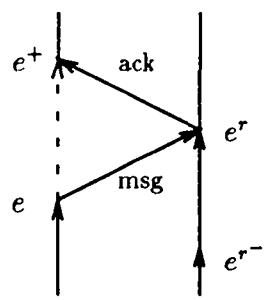

Figure 12: Exécution synchronisée par RPC

De tels systèmes présentent deux avantages pour nous. D'une part on peut effectivement calculer une estampille $\delta(e)^-$ qui caractérise les ordres d'intervalles, et d'autre part la deuxième estampille $\delta(e)^+$ est connue sur le site où e se produit, éventuellement avec un temps de retard.

Effectivement, nous avons vu dans le précédent paragraphe que si e est une réception de message alors nous avons besoin de connaître la relation que existe entre e^- et e^e pour calculer $|Pred_{\mathcal{E}}|$. Dans le cas des RPC nous avons le résultat suivant.

Proposition 3 *Soit $\mathcal{E} = (E, \theta)$ l'ordre de causalité d'une exécution répartie avec communication par RPC. Si $\mathcal{E}$ est un ordre d'intervalles, alors l'ensemble d'intervalles $([\delta(e)^-, \delta(e)^+[)_{e \in E}$ où :*

$$\delta(e)^- = \begin{cases} \delta(e^e)^- + 1 & \text{si } e \text{ est une réception d'accusé} \\ max(\delta(e^-)^-, \delta(e^e)^-) + 2 & \text{si } e \text{ est une réception de message} \\ \delta(e^-) + 1 & \text{sinon} \end{cases}$$

$$\delta(e)^+ = min\{\delta(f)^- / \theta_{im}(e, f)\}$$

est une représentation intervallaire de $\mathcal{E}$.

Preuve Nous montrons que la valeur de $\delta(e^-)$ vérifie l'hypothèse du théorème 5, à savoir :

$$\forall x, y \in E, \quad \delta(x)^- \leq \delta(y)^- \Longrightarrow Pred_{\mathcal{E}}(x) \subseteq Pred_{\mathcal{E}}(y).$$

Pour cela nous montrons que cela correspond au calcul de $|Pred_{\mathcal{E}}(e)|$ sauf dans un cas, mais que cela n'a pas d'incidence sur la propriété à vérifier.

Comme précédemment si e est un événement interne ou l'émission d'un message, alors $|Pred_{\mathcal{E}}(e)| = |Pred_{\mathcal{E}}(e^-)| + 1$.

Si E est la réception d'un accusé de réception alors on a la relation
$\theta(e^-, e^e)$ et ainsi $Pred_{\mathcal{E}}(e) = Pred_{\mathcal{E}}(e^e) \cup \{e^e\}$.
D'où on sait calculer $|Pred_{\mathcal{E}}(e)| = |Pred_{\mathcal{E}}(e^e)| + 1$.

Si e est la réception d'un message, comme $\mathcal{E}$ est ordre d'intervalles, ses ensembles de prédécesseurs sont inclus les uns dans les autres, on a :
$|Pred_{\mathcal{E}}(e)| = max(|Pred_{\mathcal{E}}(e^-)|, |Pred_{\mathcal{E}}(e^e)|) + (1 \text{ or } 2)$.
Ainsi si on effectue
$\delta(e)^- = max(\delta(e^-)^-, \delta(e^e)^-) + 2$ il se peut que l'on compte un des événements e^e ou e^- deux fois. Donc on peut dire qu'on a la relation :
$\delta(e)^- = |Pred_{\mathcal{E}}(e)| + K(e)$ où $K(e)$ correspond au nombre d'événements d'émission précédant e où on a compté un événement deux fois.

Soit e une émission de message, si e est compté une seule fois (respectivement deux fois) quand on calcule $\delta(e^r)^-$ alors, grâce au message de retour, il est aussi compté une seule fois (respectivement deux fois) quand on calcule $\delta(e^+)^-$. Ainsi les estampilles de tous les successeurs de e comptent e toujours une fois ou toujours deux fois. Cela tient au message de retour systématique. Ainsi on peut dire que $K(e)$ est une fonction croissante de $Pred_\mathcal{E}(e)$, et on a bien :

$$\begin{array}{lll} \forall e, f \in E, \quad Pred_\mathcal{E}(f) \subset Pred_\mathcal{E}(e) & \Longrightarrow K(f) \leq K(e) \\ & \Longrightarrow \delta(f)^- < \delta(e)^- \\ \text{c'est à dire} \quad \delta(e)^- \leq \delta(f)^- & \Longrightarrow Pred_\mathcal{E}(e) \subseteq Pred_\mathcal{E}(f) \end{array}$$

car tous les ensembles de prédécesseurs sont ordonnés par l'inclusion.

□

Le calcul de la seconde estampille $\delta(e)^+$ est aussi facilité.

- si e est un événement interne ou la réception d'un accusé de réception alors $\delta(e)^+ = \delta(e^+)^-$ et cette valeur est connue avec un retard.
- Si e est l'émission d'un message alors $\theta(e^r, e^+)$ and $\delta(e)^+ = \delta(e^r)^-$. Ces valeurs sont connues sur le même site lors de la réception du message de retour : $\delta(e)^+ = \delta(e^+)^- - 1$.
- Si e est l'émission d'un accusé de réception alors $\delta(e^r)^- = \delta(e)^- + 1$ et nécessairement $\delta(e)^+ = \delta(e)^- + 1$, cette valeur est connue lorsque e se produit.

Dans le cadre des exécutions synchronisées, on est donc en mesure de produire une extension intervallaire optimale dans le sens où elle caractérise les ordres d'intervalles. On décrit l'algorithme correspondant.

Algorithme d'estampillage

```
h := 0;          (* horloge logique *)
t := 0;          (* valeur précédente de h *)
e⁻ := "";        (* nom de l'événement précédent *)
s := faux;       (* vrai si l'événement précédent est un accusé *)

A chaque événement e ∈ E faire début
    t := h;
    cas
        e est un événement interne alors
            h := h+1;
            si ¬s et e⁻ ∈ O alors  trace(e⁻, t, h);
            s := faux;
        e est l'émission d'un message m alors
            h := h+1;
            si  ¬s et e⁻ ∈ O alors  trace(e⁻, t, h);
            s := faux;
            !m(h); (* émission de m estampillé par h*)
        e est la réception d'un message m estampillé par h' alors
            h := max(t, h')+2;
            si ¬s et e⁻ ∈ O alors trace(e⁻, t, h);
            si e∈ O alors trace(e, h, h+1)
            s := true;
            !ack(h); (* émission de l'accusé estampillé par h *)
        e est la réception d'un accusé estampillé par h' alors
            h := h'+1;
            si e⁻ ∈ O alors trace(e⁻, t, h')
            s := faux;
    fin
    e := e⁻;
fin
```

5 Conclusion et perspectives

Tout au long de cet article nous avons voulu montrer que la théorie des ordres partiels pouvait aider à l'élaboration d'outils d'observation et d'analyse d'exécutions distribuées. En effet il y a un écart important entre les deux seules méthodes d'estampillage connues : celle proposée par Lamport en 1978 est efficace mais imprécise tandis que celle proposée par Fidge et Mattern dix ans plus tard observe exactement l'ordre des événements mais son usage n'est pas réaliste.

Pour chercher des estampillages intermédiaires, nous nous sommes servi de la classe des ordres d'intervalles et nous avons mis en évidence une nouvelle technique d'estampillage. Nous

ordres d'intervalles et nous avons mis en évidence une nouvelle technique d'estampillage. Nous avons montré qu'elle était particulièrement adaptée pour les systèmes où la communication se fait par RPC.

Il semble donc intéressant de chercher d'autres classes d'ordres pouvant être codés avec des vecteurs de taille constante. De même, il est intéressant d'étudier quelle est l'influence d'un protocole, d'un mode de communication ou de la topologie d'un réseau sur les propriétés de l'ordre de causalité. Cette étude doit pouvoir mener à de nouveaux algorithmes d'estampillage calculant des approximations de l'ordre de causalité.

Références

[Cha89] B. Charron-Bost. *Mesures de la concurrence et du parallélisme des calculs répartis.* thèse, Université de Paris VII, 1989.

[Cha91] B. Charron-Bost. Concerning the size of logical clocks in distributed systems. *Information Processing Letters*, 39(1):11-16, july 1991.

[CKS86] W.A. Cook, M. Kress, and M. Seiford. An axiomatic approach to distance on partial orderings. *R.A.I.R.O. Recherche opérationnelle/Operations Research*, 20(2):115-122, Mai 1986.

[Dil50] R.P Dilworth. A decomposition theorem for partially ordered sets. *Annals of Math.*, 51:161-165, 1950.

[Fid88] J. Fidge. Timestamps in message passing systems that preserve the partial ordering. In *Proc.* 11th *Australian Computer Science Conference*, pages 55-66, 1988.

[Fis85] P.C Fishburn. *Interval orders and interval graphs.* Wiley, New York, 1985.

[Jez89] J.M. Jézéquel. Building a global time on parallel machines. In *Proc. of the* 3rd *International Workshop on Distributed Algorithms*, pages 136-147, Lecture Notes in Computer Science, Springer Verlag, 1989. ou Outils pour l'expérimentation d'algorithmes distribués sur machines parallèles, Thèse, Université de Rennes I, 1989.

[Lam78] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558-565, July 1978.

[Mat89] F. Mattern. Virtual time and global states of distributed systems. In Cosnard, Quinton, Raynal, and Robert, editors, *Proc. Int. Workshop on Parallel and Distributed Algorithms Bonas, France, Oct. 1988*, North Holland, 1989.

[Moh89] R.H. Möhring. Computationally tractable classes of ordered sets. In *Algorithms and Order*, pages 105-193, Kluwer Academic Publishers, 1989.

[PY79] C. H. Papadimitriou and M. Yannakakis. Scheduling interval-ordered task. *Siam J. Comput.*, 8(3):405-409, August 1979.

[Ram91] M. Habib, M. Morvan and J.X. Rampon. Remarks on some concurrency measures. In Graph–Theoretic Concepts in Computer Science, LNCS 484 :221–238, June 1990. ou J.X. Rampon. Mesures de concurrence et extensions d'intervalles. Thèse, Université de Montpellier II, 1991.

[Ray91] M. Raynal. *La communication et le temps dans les réseaux et les systèmes répartis. Collection EDF*, Eyrolles, 1991. 230 p.

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

PI 610 SYNCHRONIZATION AND CONCURRENCY MEASURES FOR DISTRIBUTED COMPUTATIONS
Michel RAYNAL
Octobre 1991, 20 pages.

PI 611 MALI v06 - TUTORIAL AND REFERENCE MANUAL
Olivier RIDOUX
Octobre 1991, 86 pages.

PI 612 SENSITIVITY COMPUTATION IN NETWORK RELIABILITY ANALYSIS
Gerardo RUBINO
Octobre 1991, 38 pages.

PI 613 OPAC : A FLOATING-POINT COPROCESSOR DEDICATED TO COMPUTE-BOUND KERNELS
André SEZNEC, Karl COURTEL
Octobre 1991, 28 pages.

PI 614 CONTROLLING AND SEQUENCING AN HEAVILY PIPELINED FLOATING-POINT OPERATOR
André SEZNEC, Karl COURTEL
Octobre 1991, 28 pages.

PI 615 ON FAULT-TOLERANT SYMBOLIC COMPUTATIONS
Bernard DELYON, Oded MALER
Novembre 1991, 18 pages.

PI 616 USING COHERENCE TO ACCELERATE RADIOSITY
Pierre TELLIER, Eric MAISEL, Kadi BOUATOUCH, Eric LANGUENOU
Novembre 1991, 16 pages.

PI 617 INTERVAL APPROXIMATIONS OF MESSAGE CAUSALITY IN DISTRIBUTED EXECUTION
Claire DIEHL, Claude JARD
Novembre 1991, 44 pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249 - 6399